Discrete Event Dynamical System: A Formulation of a Simulation Modelling(Complex Systems with Decision Makers)

AUTHOR(S):

SATO, Ryo

CITATION:

SATO, Ryo. Discrete Event Dynamical System: A Formulation of a Simulation Modelling(Complex Systems with Decision Makers). 数理解析研究所講究録 1992, 809: 197-212

ISSUE DATE:

1992-09

URL:

http://hdl.handle.net/2433/82979

数理解析研究所講究録
第 809 巻 1992 年 197-212

Discrete Event Dynamical System: A Formulation of a Simulation Modelling

Ryo SATO
Institute of Socio-Economic Planning,
University of Tsukuba,
Tsukuba city, Ibaraki 305, Japan

1. Introduction

There are three issues in decision making based on simulation: modelling, programming, and statistics(Fishman 1973). This paper concerns modelling and programming.

By simulation we mean here that of queuing structure and called discrete event simulation. Although calculation of a system of difference equations is sometime called as simulation, we restrict the usage of the word. That kind of calculation should be taken as discrete time system. Relation between discrete event systems and discrete time systems has not been much clear yet.

Pidd[2] has noticed that there are three types of errors on validation of simulation models after considering what is meant by validation concluding that a model can be said to be valid for particular purpose under specific assumptions. The first and second ones are well known in statistical hypothesis testing. A Type I error occurs when an valid model is wrongly rejected. An error of Type II occurs when an invalid model is taken to be valid. Much more severe is of Type Zero. It occurs modeler/tester asks the wrong questions, and then the model resulted is over-elaborated and/or over-simplified. It can be happen that a part of the model is too elaborated while other part has insufficient detail. To avid errors of this type Pidd[2] has proposed involvement of client of the study and the users of the result to use their knowledge through usage of non-technical language, diagrams and/or graphics to represent the problems and models. Furthermore he insisted that explicit and evolutionary approach to models and programs should be used because it is better to start a simple skeleton model than to have over-elaborated disaster that no one can understands.

To be realize the above idea Pidd[2] proposed a simple diagram, called an activity cycle diagram, for provision of non-technical representation of a model, and three phase simulation program which is supposed to be carry out the simulation according to the interaction of entities' behavior that is depicted in the diagram. (According to Pidd[2], activity cycle diagrams were popularized by Hills (Hills, P.R., *HOCUS*, P.E. Group, Egham, 1971) and three phase approach to discrete event simulation was first described by Tocher (Tocher, K.D., *The Art of Simulation*, English Universities Press, 1963)). Although it seems to be easy to understand and do actually the methodology, by which one can start to make an activity cycle diagram and then

build a corresponding program to get data of simulation experiments, several important issues are not much clarified.

In this paper the methodology proposed by Pidd[2] as three phase approach, which is shown in Fig.1, is formulated in terms of dynamics and interaction within the modelling process. That is, a state space representation of a discrete event system is constructed. The formulation is to provide systems theoretical basis of simulation modelling and then to give insight for avoiding errors of Type Zero. The questions we would like to answer are:

(i) What is a discrete event system?

(ii) What information fully decides the dynamics of a discrete event system? That is, in rigorous sense what kind of dynamics is dealt with and how?

(iii) How interactions are modelled and built into a program?

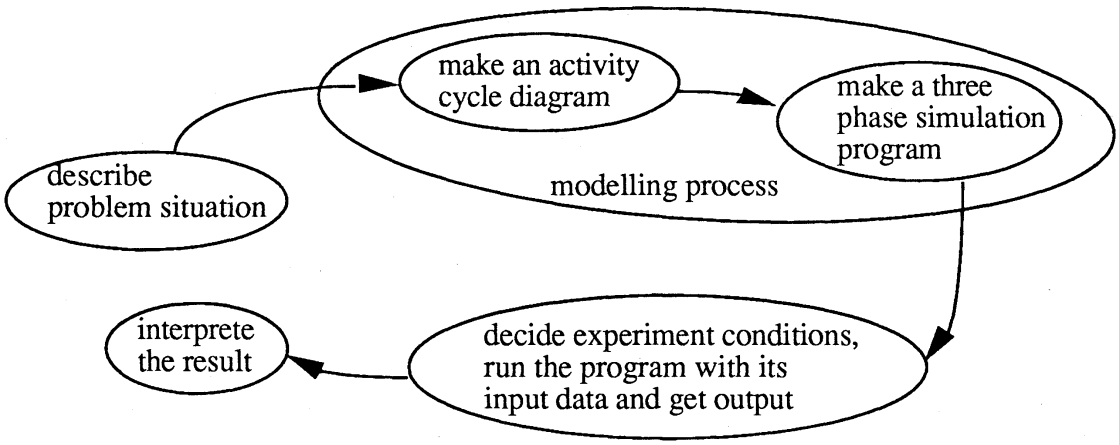

Fig. 1. Outline of Simulation Methodology

2. Basic Concepts

In this section systems, state space representations and some notation are defined according to Mesarovic and Takahara[1]. State space is a key concept by which we can grasp the behavior of the dynamic system in causal way.

Definition 1. *system*

A system is a relation of an input set and output set. If those two sets are sets of time functions defined on the same time set, then the system is called a time system.

The value set of inputs of a time system is called input alphabet and that of output called output alphabet. Usually one of the set of non-negative real numbers R^+ or that of non-negative integers Z^+ is taken as a time set. Let x be a function from a time set T to a input alphabet. For any $t, t' \in T$, $t \leq t'$, the restriction of domain of x to $[t, t')$ is written as $x_{tt'}$. That is, $x_{tt'}(\tau) = x(\tau)$

for any τ, $t \le \tau < t'$. Similarly x_t is defined as $x_t(\tau) = x(\tau)$ for any τ, $t \le \tau$. Let x and x' be arbitrary functions from T to the same alphabet A. For any $t \varepsilon T$, we can define another element $x'': T \to A$ by

$$x''(t'') = \begin{cases} x(t''), & \text{if } t'' < t \\ x'(t''), & \text{if } t'' \ge t \end{cases}$$

x'' is called the concatenation of x_{0t} and x'_t and denoted by $x'' = x_{0t} \bullet x'_t$.

Definition 2. *state space representation*

Let $S \subset X \times Y$ be a time system with output alphabet B. A pair $<\Phi, \mu>$ where

$$\Phi = \{\phi_{tt'} \mid \phi_{tt'} : C \times X_{tt'} \to C \text{ and } t, t' \varepsilon T, t \le t'\}$$

and $\mu : C \to B$

is a state space representation of S if and only if the following conditions are satisfied:

(i) the functions Φ satisfies the following

(α) $\phi_{tt''}(c, x_{tt''}) = \phi_{t't''}(\phi_{tt'}(c, x_{tt'}), x_{t't''})$, where $t \le t'' \le t'$ and $x_{tt''} = x_{tt'} \bullet x_{t't''}$

(β) $\phi_{tt}(c, x_{tt}) = c$

(ii) $(x,y) \varepsilon S$ if and only if there exists some $c \varepsilon C$ such that for any $t \varepsilon T$

$$y(t) = \mu(\phi_{0t}(c, x_{0t})).$$

C is called the state space of $<\Phi, \mu>$.

Especially Φ is called a transition family if it satisfies (α) and (β) of the above definition.

State space representations are wide-spread framework to recognize dynamics of a time system in causal way. Mesarovic and Takahara[1] shows that a time system is causal if and only if it has a state space representation. Therefore what decides the dynamics of a system is equivalent to what information can be used as a state space.

3. Discrete Event System

The target of our study, discrete event system is defined as follows:

Definition 3. *discrete event system*

If a time system $S \subset X \times Y$ satisfies the following four conditions then is referred to be as discrete event system:

1) $T=[0, T_{end})$, $T_{end} \varepsilon R^+$

2) There is a set A' and the input alphabet A of input space X is a power set of A'. That is, $A=P(A')$, where P(A') is the power set of A'.

3) For any $x \varepsilon X$, $\{t \mid x(t) \neq 0$, 0 is the empty set$\}$ is finite set

4) For any t_1 and t_2, $t_1 < t_2$, and any $(x,y) \varepsilon S$, if $y(t_2) \neq y(t_1)$ then $x(t) \neq 0$ holds for some t, $t_1 < t \leq t_2$.

In the above definition the third character is representation of a "discrete event" system. Taking an element of the set A' as an activity, the second character shows parallel execution of some activities. The fourth character represents that if there happens no event then the corresponding output remains at the same value. In other words the concerned system has no variable that changes continuously with time. A discrete event system is specified by a sextuple (A', B, T, X, Y, S). When there is no confusion we simply call $S \subset X \times Y$ a discrete dynamical system.

For an input x the set $\{t \mid x(t) \neq 0, 0$ is the empty set$\}$ is denoted by event(x) and an element of event(x) is called an event of x. Since event(x) is finite for any input x, we can think that event(x) = $\{t_1, t_2, ..., t_n\}$, $t_1 < t_2 < ... < t_n$ for some integer n. A function nextevent : $X \rightarrow \{T \rightarrow T\}$ is defined as

$$\text{nextevent}(x)(t) = \begin{cases} t_k, \text{ if } t_{k-1} \leq t < t_k \text{ for some } k, 1 \leq k \leq n \\ T_{end}, \text{ if } t_n \leq t \end{cases}$$

,where $t_0 = t$. As its name shows the nextevent function shows the next event of x at t.

In order to establish a realization theory which can be directly implemented as a computer program for simulation of a discrete event system, we need the following transformation of inputs of discrete event systems.

Definition 4. *L (time-list representation)*

Let x be an input of a discrete event system (A', B, T, X, Y, S). Notice $x(t) \varepsilon P(A')$ holds for any $t \varepsilon T$.

Define $L: X \rightarrow \{T \rightarrow \{A' \rightarrow R^+\}\}$ as

$$L(x)(t)(b) = \begin{cases} T_{end}, \text{ if } x(t') \text{ does not include b for any } t', t \leq t' < T_{end}, \\ \min\{t' \mid x(t') \text{ includes b at } t', t' > t\}, \text{ otherwise} \end{cases}$$

for any $x \varepsilon X$, $t \varepsilon T$, and $b \varepsilon A'$. L(x) is called a time-list representation of x.

For any input function x, L(x) is referred to as time-list representation (or simply time-list) of x. The meaning of L(x) is simple and illustrated in Figure 1. Any input x of any discrete event system has finite events(x) by definition 3. So event(x) can be written as $\{t_1, t_2, ..., t_n\}$, and without loss of generality $t_i < t_j$ holds if $i < j$. When we write L(x) as x^, x^(t) is a function from A' to R^+ for any time t. Let b be an event. Then the fact x^(t)(b) = t_3, for example, shows that at the time t the event b will occur at t_3. Since A' is finite, x^(t) can be seen as a table or a vector which tells each event will happen at each of recorded times.

The following lemma shows some relation between L(x) and events of x.

Lemma 1

Let x be an arbitrary input of a discrete event system and event(x) = $\{t_1, t_2, ..., t_n\}$, $t_1 < t_2 < ... < t_n$. Then for any k, $1 \leq k-1 \leq n-1$, the followings hold:

(i) $L(x)_{tt'}(\tau) = L(x)(t_{k-1})$

for any t, t', $t_{k-1} \leq t < t' \leq t_k$, and any τ, $t \leq \tau < t'$.

(ii) $L(x)_{t_{k-1}t}(\tau)(b) \geq \text{nextevent}(x)(t_{k-1}) = t_k$

holds for any $b \varepsilon A'$, any t, $t > t_{k-1}$, and any τ, $t_k \leq \tau < t'$. Furthermore there is a $b' \varepsilon A'$ such that $L(x)_{t_{k-1}t}(\tau)(b') = t_k$.

Proof. It is clear from the above definitions of L and nextevent. □

The function L has its inverse which is written as L^{-1}. For a discrete event system S whose input space is X, the time system whose input space is L(X) is written as L(S).

Definition 5. *discrete event dynamical system*

Let S be a discrete event system. A discrete event dynamical system of S is a state space representation of L(S).

The reason why the input of a state space representation of a discrete event system is L(X) instead of X, is that the formulation is in programming orientation. In Section 5 a simulation program will be formulated and its input is L(X). Since it is shown that the program is a state space representation and then we can get a concrete program of the dynamics. In other words this definition provides us not only an analysis of dynamics of discrete event systems but also the way how to implement it as a program.

4. Activity Interaction Diagram

As depicted in Fig. 1, Pidd's approach to simulation modelling is two fold. The first is creation of diagrammatic skeleton model called an activity cycle diagram. The concept of activity cycle diagram is modified and then the name is changed as activity interaction diagram. The reason of the change is explained later.

Definition 6. activity interaction diagram

A seventuple (E, EC, A, A', W, D, f_{EA}) is called an activity interaction diagram if it satisfies following conditions;

1) E is a finite set named as entity set and its element is called as entity. An equivalence class is defined on E and the quotient set is EC. An equivalence class of entity set is called as an entity class.

2) A is a finite set named as activity set and its element is called as activity. Each entity class corresponds to activities which is specified by the function f_{EA}:EC $\rightarrow$ P(A).

3) W is a finite set named as wait set and its element is called a waiting queue.

4) Let V=A $\cup$ W. The graph D $\subset$ V×V satisfies the following three conditions:

4-1) For any a ε A, (a, w) ε D holds for some w ε W.

4-2) (s_1, s_2) ε D $\rightarrow$ (s_1 ε A & s_2 ε W) or (s_1 ε W & s_2 ε A): alternation of A and D.

4-3) D is finite.

5) A' = { a ε A | (w, a)$\notin$ D holds for any w ε W} $\neq$ 0, where 0 is the empty set.

An activity interaction diagram is a graph. Fig. 2 depicts an example of an activity interaction diagram. $s_1 \rightarrow s_2$ in graph representation is written as (s_1, s_2) in definition 6. A path of arrows in accordance with their directions shows how an entity of an entity class behaves through time.

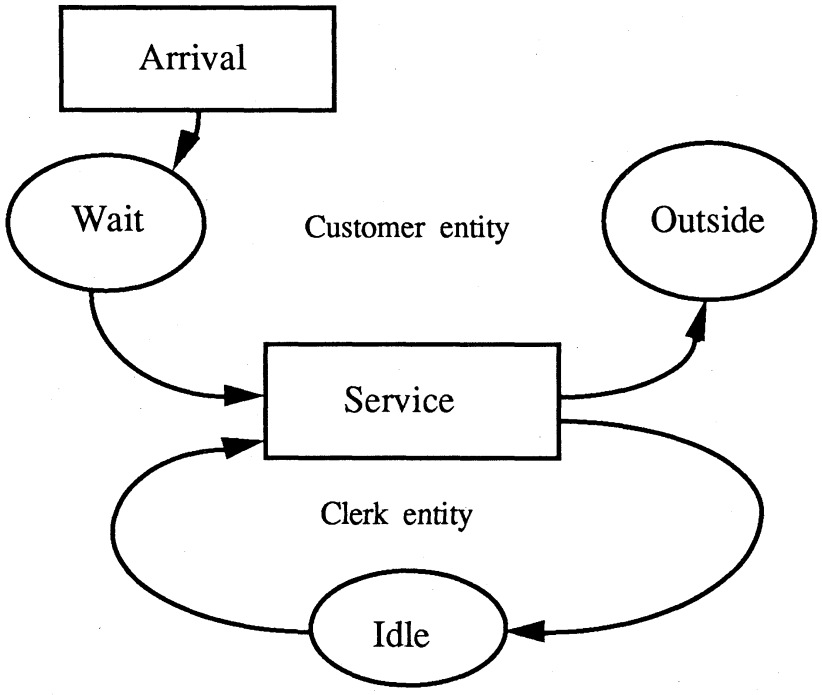

Fig. 2. acitivity interaction diagram

An activity is also called activity state and represented by a square, and a waiting queue called also wait state is by an ellipse. All of waiting queue are not real queue in a simulation program. Some of them are used just to represent conditions by which an activity can occur. In such case both 0 (or false) and 1 (or true) might be used as possible values of the queue. Before

some activity starts the previous activity must be finished and an entity must be in a certain wait state. Thus activity state and wait state appear alternatively in any path.

An element of A' is called B activity. "Arrival" in Fig. 2 is an example. In the simulation program any B activity bootstrapped to decide the next occurrence time when it occurs. An element of A, that is not in A', is called C activity. Its occurrence is conditioned by related queues in the program. An example of C activity is "Service" in Fig. 2.

Activity interaction diagrams are slightly different from activity cycle diagrams which is defined in Pidd[2]. Firstly the name is different, and secondly any possible path with respect to an entity may not make closed cycle in activity interaction diagram while activity cycle diagram urges to do that.

The name is changed because one of the main information represented by the diagram is interactions between entities' behavior as Pidd himself said. Making cycle has no effect in making a corresponding simulation program and no conceptual value. Precisely speaking, interaction between entities is depicted through a typical (or representative) entity of the same character, that is, through classes of entities. The interaction suggests the existence of conditions on mutual possible behavior of entities of different classes of entities at any time.

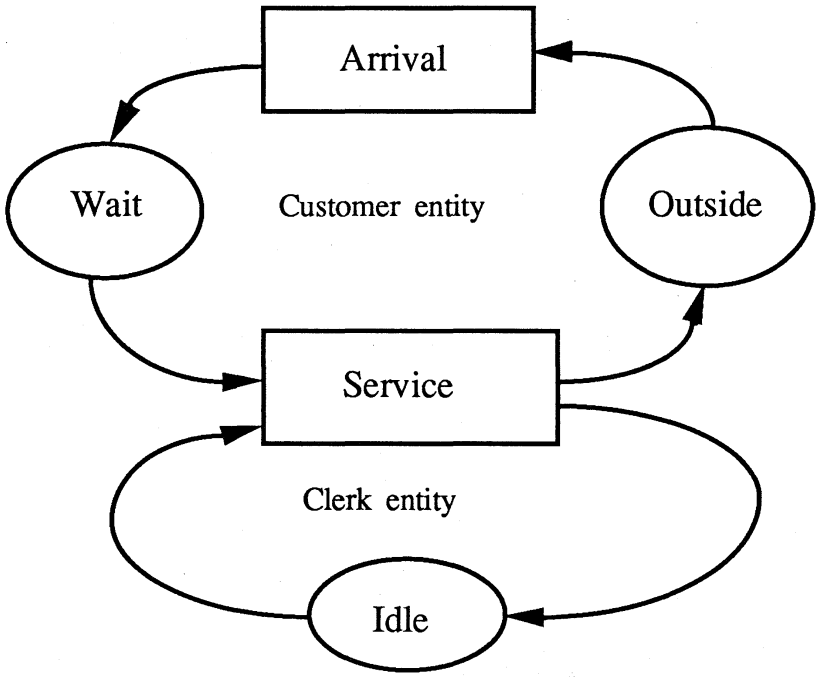

Fig. 3. acitivity cycle diagram

Although both activity cycle diagrams and activity interaction diagrams have the same philosophy, the former urged to make any path circle. For example, the same interaction of clerk and customer in a fast food restaurant depicted in Fig.2 is shown as activity cycle diagram in

Fig.3. In modelling a problem situation by activity interaction diagram modelers think about history of not a class of entities, but a typical entity, like a customer or a clerk. An arrow from "Outside" to "Arrival" not only does not be needed but also can be misleading because a customer entity may not come again to the window after once got the service. Furthermore other customers can arrive while the previously arrived customer is still in queue, not outside. Thus when there are activities, like arrival, that occur independently from other status of the system, cycle representation does not seem to be suitable.

The above two kind of consideration make us change the name and definition of activity cycle diagram as activity interaction diagram.

An activity interaction diagram can correspond to discrete event system. One of them is defined below.

Definition 7. *consistent discrete event system*

Let (A'_S, B, T, X, Y, S) be a discrete event system. It is said to be consistent with a activity interaction diagram $(E, EC, A, A', W, D, f_{EA})$ if satisfies the following conditions:

1) $A'_S = A'$.

2) The alphabet of output Y is the set of functions $\{f \mid f:W \rightarrow Z^+\}$,

3) Let $(x, y) \varepsilon S$, $w \varepsilon W$, and $t_1, t_2 \varepsilon T$ be arbitrary. Assume $t_1 < t_2$ holds.

Then there exist $a \varepsilon A$, $(a, w) \varepsilon D$, and $t \varepsilon T$, $t_1 < t \leq t_2$, such that $[y(t_1)(w) \neq y(t_2)(w) \rightarrow a \varepsilon x(t)]$ holds.

5. Three Phase Simulation System

In the following definition a framework of simulation program based on the three phase approach[2] is formulated. And it will be proved to be a state space representation (discrete event dynamical system) of a discrete event system. In the following /*...*/ is a comment.

Definition 8. *three phase simulation system* $<\Phi, I_Q>$

8-1) definition of auxiliary sets

Clock = $[0, T_{end}]$, where $T_{end} \varepsilon R^+$ /* time set of three phase simulation system */

$\Delta \varepsilon$ Clock /* The smallest time-slice to proceed simulation */

Entities = {1, 2, ..., n} /*Every entity has a unique name as a number*/

EntityClass = Entities/≡ , where ≡ is an equivalence relation on Entity

AnEntityState = TimeCell×NextActivity×Avail

,where TimeCell = Clock

NextActivity = ActivitySet = BActivitySet ∪ CActivitySet

/* BActivitySet and CActivitySet are finite */

Avail = {available, void} /*shows whether the time cell of the entity state is

currently available or not*/

/* Each entity is associated to an activity. If the activity is an element of BActivitySet and the value of Avail is "available" then the activity will occur at the time represented in TimeCell. If the activity is of CActivitySet possibility of occurrence of it is examined at each time any BActivity and CActivity occurred. */

EntityStates: Entities → AnEntityState /* entity has its state */
QueueId is a finite set. /* each queue in the system has its name as a number */
Queue: QueueId → Number /*Queue shows the length of a queueid */
f_{AfectQ}: ActivitySet → QueueId /* Each activity affects a queue specified by f_{AfectQ}. */
X = {x | x : Clock → P(BActivitySet), {t | x(t)≠0, 0 is the empty set} is finite} /* input */
BTL = {f | f : BActivity → $[0, T_{end}]$} /*Note that L(x)(t) is an element of BTL for any
$t \varepsilon [0, T_{end}]$ */

/* If z is an element of a Cartesian product and has A-coordinate then the A-coordinate of z is written as "z.A". For example, if z = (a, b, c) ε A×B×C then z.A = a and z.B = b. The chlonological change of EntityStates and Queue can be divided into three phase, each of which is named A_Phase, B_Phase, and C_Phase. */

8-2) Transition in A_Phase

/* The transition of this phase is time scan and characterized by functions f_{Scan} and f_{Dues}. */

f_{Scan}: EntityStates → Clock
f_{Scan}(entitystates)
= min {k | k=entitystates(i).TimeCell and
entitystates(i).Avail=available for some entity i ε Entity}
f_{Dues}: EntityStates → P(Entities), where P(Entities) is the set of subsets of Entities.
f_{Dues}(entitystates) ={i | f_{Scan}(entitystates)=entitystates(i).TimeCell,
and entitystates(i).Avail = available}

8-3) Transition in B_Phase

/* The transition of B_Phase is characterized by functions f_{Type}, f_{AfectQ}, f_{B_Ent}, and f_{B_Que}. */
f_{Type}: ActivitySet → {ScheduleNext, NonSchedule}

/* This indicates whether a BActivity should be 'bootstrapped' or not. If an activity is bootstrap type, the TimeCell value is reassigned when it occurs. And that TimeCell value shows when the activity occurs next time.*/

We define f_{B_Ent} as follows.
Let entitystates ε EntityStates and x be arbitrary, nextdues = f_{Dues}(entitystates) and c = f_{Scan}(entitystates).
f_{B_Ent} : Entities×EntityStates×BTL → AnEntityState
f_{B_Ent}(i, entitystates, β) = entitystate such that
case1: when i is not in nextdues:
entitystate = entitystates(i) /* unchanged*/
case2: when i is in nextdues :
case2.1: when f_{Type}(entitystates(i).NextActivity) = NonSchedule:
entitystate.NextActivity = entitystates(i).NextActivity /*unchanged*/
entitystate.TimeCell = entitystates(i).TimeCell /* unchanged*/
entitystate.Avail = void.
case2.2: when f_{Type}(entitystates(i).NextActivity) = ScheduleNext:
entitystate.NextActivity = entitystates(i).NextActivity /* unchanged*/
entitystate.Avail = available
entitystate.TimeCell = β(entitystates(i).NextActivity) /* "bootstrap" */
,where β(entitystates(i).NextActivity) ≥ Δ + c.

f_{B_Que}: Entities×EntityStates×Queue → Queue
/* f_{B_Que} calculates Queue when any of BActivity occurs. */
f_{B_Que}(i, entitystates, queue) = queue' such that

$$\text{queue}'(k) = \begin{cases} f_{QueVal}(\text{entitystates(i).NextActivity, queue}), \\ \quad \text{if } i \in \text{nextdues and } k \in f_{AfectQ}(\text{entitystates(i).NextActivity}) \\ \text{queue(k)}, \text{ otherwise}, \end{cases}$$

where f_{QueVal}:ActivitySet×Queue → Number

/* Each activity is associated to a queue specified by fAfectQ. If an activity is supposed to take place the associated queues are calculated by fB_Que. Whether an activity occurs or not at the time c is decided from the fact that the activity is an element of nextdues. */

8-4) transition in C_Phase

$f_{C\_condition}$: Queue → Entity ∪ 0, where 0 represents the empty set.

/* Based on the status of the whole queue this function decides which CActivities occur by specifying entities whose NextActivity is an CActivity. */

Let i be an arbitrary element of Entity.

f_{C_Ent}:Entities×EntityStates → AnEntityState

f_{C_Ent}(i, entitystates) = entitystate such that

case1: when i is not in $f_{C\_condition}$(queue):

entitystate = entitystates(i) /* unchanged */

case2: when i is in $f_{C\_condition}$(queue):

entitystate.NextActivity =$f_{CBNextAct}$(entitystates(i).NextActivity)

entitystate.TimeCell = $f_{NextTime}$($f_{CBNextAct}$(entitystates(i).NextActivity)),

entitystate.Avail = available

,where $f_{CBNextAct}$ is a function $f_{CBNextAct}$: CActivitySet → BActivity, and

$f_{NextTime}$: Clock → BTL such that $f_{NextTime}$($f_{CBNextAct}$(entitystates(i).NextActivity)) ≥ Δ + c.

/* There are two types of BActivities. One is bootstrapping type. The other is non bootstrapping and set into an entity to occur sometime by a CActivity. For example, assume that there is a sales clerk at ticket office and she accepts phone call if there is no customer in the office. The start of phone conversation is a CActivity because it happens when both there is no customer and there is a phone call. If she starts phone call service the time to finish it is solely determined. That is, the activity "end of phone call" is a BActivity of non bootstrapping. */

f_{C_Que}: Entities×EntityStates×Queue → Queue

/* f_{C_Que} calculates Queue when any of CActivity occurs. */

f_{C_Que}(i, entitystates, queue) = queue' such that

$$\text{queue}'(k) = \begin{cases} f_{QueVal}(\text{entitystates(i).NextActivity, queue}), \\ \quad \text{if } i \in f_{C\_condition}(\text{queue}) \text{ and } k \in f_{AfectQ}(\text{entitystates(i).NextActivity}) \\ \text{queue}(k), \text{ otherwise.} \end{cases}$$

8-5) construction of $<\Phi, I_Q>$

f_B:EntityStates×Queue×BTL → EntityStates×Queue.

$f_B(e, q, \beta)=(e', q')$ such that

$e' = (f_{B\_Ent}(1, e, \beta), f_{B\_Ent}(2, e, \beta), ..., f_{B\_Ent}(n, e, \beta)$, and

$q' = q'_n$, where $q'_0 = q$, $q'_k = f_{B\_Que}(k, e, q'_{k-1})$ for any k, $1 \le k \le n$.

/*f_{Scan} and f_{Dues} are used in calculation of f_B. */

/* f_C is a total function which satisfies the following */

f_C : EntityStates×Queue → EntityStates×Queue.

$$\begin{cases} (e, q), \text{ if } f_{C\_condition}(q) = 0, \end{cases}$$

$$f_C(e, q)= \begin{cases} \\ f_C(f_{C\_Ent}(e), f_{C\_Que}(e, q)), \text{ if } f_{C\_condition}(q) \neq 0, \end{cases}$$

where $f_{C\_Ent}(e) = (f_{C\_Ent}(1, e), f_{C\_Ent}(2, e), ..., f_{C\_Ent}(n, e))$, and

$f_{C\_Que}(e,q) = q'_n$, where $q'_0 = q$, $q'_k = f_{C\_Que}(k, e, q'_{k-1})$ for any k, $1 \leq k \leq n$.

δ:EntityStates×Queue×BTL $\rightarrow$ EntityStates×Queue

$\delta = f_C \bullet f_B$

f_{EBact}:EntityClass $\rightarrow$ BActivitySet /* Every entity class corresponds to a B activity which is specified by f_{EBact} */

f_{XEnt}:BTL $\rightarrow$ EntityStates

$f_{XEnt}(\beta)$ = e such that

e(i).NextActivity = $f_{EBact}([i])$,

e(i).TimeCell = $\beta(f_{EBact}([i]))$,

e(i).Avail = available

for any i ε Entities, where [i] is the equivalence class that has i as its representative and $\beta(f_{EBact}([i])) \geq D + f_{Scan}(e)$.

For any t and t", t < t", $\phi_{tt"}$ is defined by

$$\phi_{tt"}(q, L(x)_{tt"})= \begin{cases} \phi_{t't"}(\delta(f_{XEnt}(L(x)_{tt"}(t)), q, L(x)_{tt"}(t')).Queue, L(x)_{t't"}), \\ \text{where } t'=\min\{t^ \mid t^=L(x)_{tt"}(t)(b) \text{ for some } b \varepsilon BActivity \text{ and } t \leq t^ \leq t'\}, \\ q, \text{ otherwise.} \end{cases}$$

Also define $\phi_{tt}(q, L(x)_{tt}) = q$ for any t.

The family of functions defined above $<\Phi, I_Q>$, where $\Phi = \{\phi_{tt'} \mid t, t' \varepsilon T, t \leq t'\}$ and I_Q is the identity function on Q, is referred to be as three phase simulation system.

In this paper we restrict our consideration in the case where case 2 in the definition of f_{C_Ent} never occurs. By this restriction we can not deal with the case that there needs some duration of a C activity. Furthermore we assume that f_C is a total function, that is, the expansion of f_C is eventually stops by $f_{C\_condition}$ value being empty. In terms of simulation programs of three phase simulation the assumption that f_C is total requires that the program always stops.

For a three phase simulation system $<\Phi, I_Q>$ the resultant system that $<\Phi, I_Q>$ defines is $Res(<\Phi, I_Q>) = \{(L(x),y) \mid (\exists q)(\forall t)\ (y(t) = \phi_{0t}\ (q, L(x)_{0t}))\ \}$.

Proposition 1

Let $<\Phi, I_Q>$ be a three phase simulation system. Φ is a transition family.

Proof. Let q ε Queue, t, t" ε T, t $\leq$ t" and x ε X be arbitrary. It will suffice to show that

(*) $\phi_{tt''}(q, L(x)_{tt''}) = \phi_{t't''}(\phi_{tt'}(q, L(x)_{tt'}), L(x)_{t't''})$

holds for any t', $t \le t' \le t''$, where $L(x)_{tt''} = L(x)_{tt'} \bullet L(x)_{t't''}$.

Since event(x) is finite, we can assume event(x) $\cap$ [t, t''] = $\{t_1, t_2, ..., t_n\}$, $t_1 < t_2 < ... < t_n$. Set $t_0 = t$ and $t_{n+1} = t''$. Then we have $t = t_0 \le t_1 < ... < t_n \le t_{n+1} = t''$. Let $q_0 = q$, and $q_k = \delta(f_{XEnt}(L(x)_{tt'}(t_{k-1})), q_{k-1}, L(x)_{tt'}(t_k))$.Queue for each k, $1 \le k \le n$. Since $\min\{t^\wedge \mid t^\wedge = L(x)_{t_{k-1}t''}(t_{k-1})(b)$ for some b ε BActivity and $t_{k-1} \le t^\wedge \le t''\} = t_k$ holds by Lemma 1, we have

(**) $\phi_{t_{k-1}t''}(q_{k-1}, L(x)_{t_{k-1}t''}) = \phi_{t_kt''}(\delta(f_{XEnt}(L(x)_{t_{k-1}t''}(t_{k-1})).Queue, q, L(x)_{t_{k-1}t''}(t_k)), L(x)_{t_kt''})$

$= \phi_{t_kt''}(q_k, L(x)_{t_kt''})$

for any k, $1 \le k \le n$.

Let t' be an arbitrary element of [t, t''). Firstly assume that $t' = t_k$ for some k, $0 \le k \le n+1$. If $t' = t_0$ or $t' = t_{n+1}$ then the equation (*) holds trivially. So assume $t' = t_k$ holds for some k, $1 \le k \le n$. Then by applying (**) k times we have

$\phi_{tt''}(q, L(x)_{tt''}) = \phi_{t_kt''}(q_k, L(x)_{t_kt''})$.

In the same way, k-1 times application of (**), where t'' is replaced by t_k, gives that

$\phi_{tt_k}(q, L(x)_{tt_k}) = \phi_{t_{k-1}t_k}(q_{k-1}, L(x)_{t_{k-1}t_k})$.

Since $\min\{t^\wedge \mid t^\wedge = L(x)_{t_{k-1}t_k}(t_{k-1})(b)$ for some b ε BActivity and $t_{k-1} \le t^\wedge \le t_k\} = t_k$, we have $\phi_{t_{k-1}t_k}(q_{k-1}, L(x)_{t_{k-1}t_k}) = \phi_{t_kt_k}(q_k, L(x)_{t_kt_k}) = q_k$. Above all we have

$\phi_{tt''}(q, L(x)_{tt''}) = \phi_{t_kt''}(\phi_{tt_k}(q, L(x)_{tt_k}), L(x)_{t_kt''}) = \phi_{t't''}(\phi_{tt'}(q, L(x)_{tt'}), L(x)_{t't''})$

, which is to be proved.

Secondly assume that $t_{k-1} < t' < t_k$ holds for some k, $1 \le k \le n+1$. By (**) we have

$\phi_{tt'}(q, L(x)_{tt'}) = \phi_{t_{k-1}t'}(q_{k-1}, L(x)_{t_{k-1}t'})$.

Since nextevent(x)(t_{k-1}) = t_k and $t_k > t'$, $\phi_{t_{k-1}t'}(q_{k-1}, L(x)_{t_{k-1}t'}) = q_{k-1}$ holds. That is, $\phi_{tt'}(q, L(x)_{tt'}) = q_{k-1}$. k-1 times application of (**) gives that

$\phi_{tt''}(q, L(x)_{tt''}) = \phi_{t_{k-1}t''}(q_{k-1}, L(x)_{t_{k-1}t''})$.

Therefore if $\phi_{t_{k-1}t''}(q_{k-1}, L(x)_{t_{k-1}t''}) = \phi_{t't''}(q_{k-1}, L(x)_{t't''})$ holds then we have

$\phi_{tt''}(q, L(x)_{tt''}) = \phi_{t't''}(q_{k-1}, L(x)_{t't''}) = \phi_{t't''}(\phi_{tt'}(q, L(x)_{tt'}), L(x)_{t't''})$

,which concludes the proof. By lemma 1 we have that $L(x)_{t_{k-1}t''}(\sigma) = L(x)_{t't''}(\tau) = L(x)(t_{k-1})$ for any σ, $t_{k-1} \le \sigma < t_k$, and any τ, $t' \le \tau < t_k$, and that $\min\{t^\wedge \mid t^\wedge = L(x)_{t't''}(t')(b)$ for some b ε BActivity and $t' \le t^\wedge \le t_k\} = t_k$. Thus we have

$\phi_{t't''}(q_{k-1}, L(x)_{t't''}) = \phi_{t_kt''}(\delta(f_{XEnt}(L(x)_{t't''}(t_{k-1})), q, L(x)_{t't''}(t_k)).Queue, L(x)_{t_kt''})$

$= \phi_{t_kt''}(q_k, L(x)_{t_kt''})$.

The equation (**) says this is equal to $\phi_{t_{k-1}t''}(q_{k-1}, L(x)_{t_{k-1}t''})$. □

Proposition 2

Let $<\Phi, I_Q>$ be a three phase simulation system. Then $L^{-1}(Res(<\Phi, I_Q>))$ is a discrete event system.

Proof. Let t_1, t_2, $t_1 < t_2$, and $(x,y) \in L^{-1}(Res(<\Phi, I_Q>))$ be arbitrary. It will suffice to show that if $x(t) = 0$ for any t, $t_1 < t \leq t_2$, then $y(t_1) = y(t_2)$. Assume that $x(t) = 0$ for any t, $t_1 < t \leq t_2$. Then there are t' and t'' such that t', t'' $\in$ event(x), nextevent(x)(t') = t'' and $t' \leq t_1 < t \leq t_2 < t''$ hold. Let $q' = \phi_{0t'}(q, L(x)_{0t'}))$. Then $\phi_{0t_1}(q, L(x)_{0t_1})) = \phi_{t't_1}(q', L(x)_{t't_1}))$. Since min { $L(x)_{t't_1}(t')(b) \mid b \in$ BActivitySet } = t'' and $t'' \notin [0,t_1]$, we have $\phi_{t't_1}(q', L(x)_{t't_1})) = q'$. Similarly we have $\phi_{0t_2}(q, L(x)_{0t_2})) = \phi_{t't_2}(q', L(x)_{t't_2})) = q'$ by noticing $t'' \notin [0,t_2]$. By definition of $L^{-1}(Res(<\Phi, I_Q>))$, $y(t_1) = \phi_{0t_1}(q, L(x)_{0t_1})) = q' = \phi_{0t_2}(q, L(x)_{0t_2})) = y(t_2)$.

□

The above two propositions provide one of the two main results.

Theorem 3

Let $<\Phi, I_Q>$ be a three phase simulation system. Then $<\Phi, I_Q>$ is a discrete event dynamical system of $L^{-1}(Res(<\Phi, I_Q>))$.

The so-called discrete event simulation is carried out by a program whose input is a time function on BActivities. Length of time between events, when several activities occur, are taken from appropriate distribution through sampling technique.

The whole simulation program works as Fig. 4. After initialization of EntityStates by assigning appropriate values, a cycle consists of A_Phase, B_Phase and C_Phase is repeated until clock exceeds the pre-determined time Tend. Every time a BActivity occurs its corresponding entitystate changes its TimeCell value by sampling from its assigned distribution. When simulation is over a pair (x, y) is provided. When much data is needed to statistical evaluation experiment design should be used, although we do not mention it in this paper.

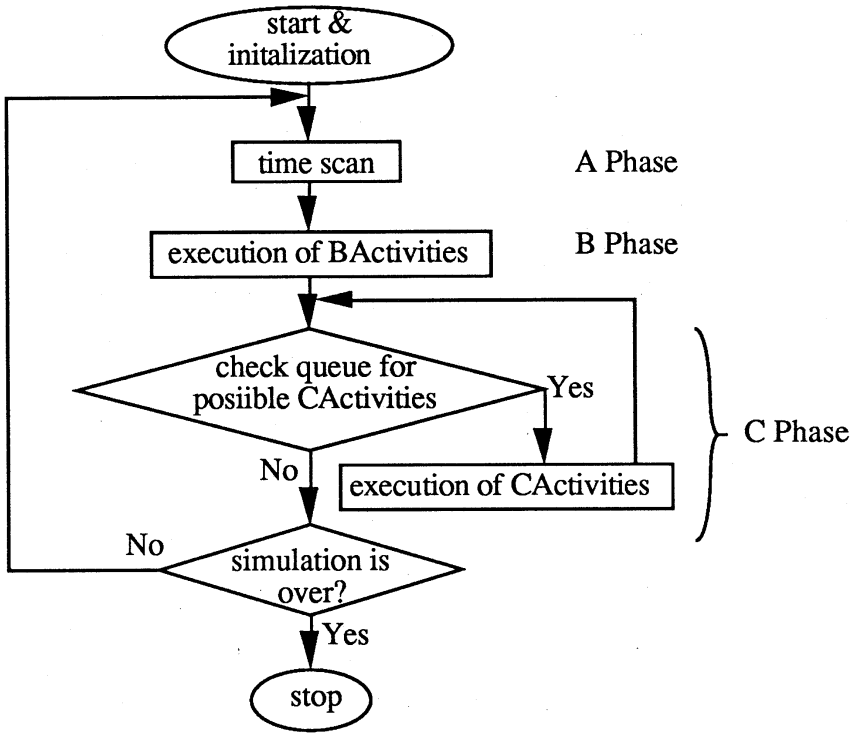

Fig. 4 . Execution of three phase simulation

Definition 9. *simulation-implementation of activity interaction diagram*

Let $(E_{aid}, EC_{aid}, A_{aid}, A'_{aid}, W_{aid}, D_{aid}, f_{EAaid})$ be a activity interaction diagram and $<\phi, I_Q>$ a three phase dynamical system. $<\phi, I_Q>$ is called a simulation-implementation of the activity interaction diagram if the following conditions hold:

1) E_{aid} = Entity, EC_{aid} = EntityClass

2) A'_{aid} = BActivitySet

3) $f_{EAaid} = f_{EBact}$

4) W_{aid} = QueueId

5) For any $a \varepsilon A'_{aid}$ = BActivitySet and $w \varepsilon W_{aid}$ = QueueId the following holds:

$f_{AfectQ}(a) = w$ if and only if $(a, w) \varepsilon D_{aid}$.

The following theorem shows how a simulation implementation of an activity interaction diagram can be made and how the information that the diagram has is used in simulation.

Theorem 4

Let $<\phi, I_Q>$ be a simulation-implementation of an activity interaction diagram. Then $L^{-1}(Res(<\phi, I_Q>))$ is consistent with the diagram.

Proof. It will suffice to show that the last condition of the definition of consistent discrete event system. Let $(x, y) \varepsilon L^{-1}(Res(<\phi, I_Q>))$, $w \varepsilon W$, and $t_1, t_2 \varepsilon T$ be arbitrary. Assume $t_1<t_2$ holds. Furthermore assume $y(t_1)(w) \neq y(t_2)(w)$. Since $L^{-1}(Res(<\phi, I_Q>))$ is a discrete event system

there is a t, $t_1 < t \leq t_2$, such that $x(t) \neq 0$. From the construction of ϕ, there must be an activity in $f_{AfectQ}^{-1}(w)$ such that queue is changed according to f_{B_Que} and/or f_{C_Que} by carrying out the activity at the event t. □

6. Conclusion

What the formulation revealed are:

i) The dynamics of three phase simulation program is formulated as a state space representation whose state space is queue (by Theorem 3), and the resultant system is a discrete event system in the defined sense (Definition 3). The question, what has the all of historical information of the system by which further dynamics of the system is fully determined, is not trivial but fundamentally important to answer what kind of system we will have by simulation.

ii) From the fact that the queue is the state space of the dynamical system, the condition check, that decides which C activities are ready to occur, should be examined about only the whole queue. There is no need to examine other information. This fact has not been exactly declared yet.

iii) The formulation of three phase simulation implementation (Definition 8) shows that the state, queue, is changed by activities and in a program (implementation) the priority of activities that can be occurred simultaneously is fixed. And the fixed priority is given by the correspondence between activities and entities.

iv) A skeleton model, activity interaction diagram, models entities, entity classes, independent activities, correspondence between activities and entities, queues and a condition which must be preserved in the simulation program of the skeleton model (Theorem 4).

As mentioned in Section 5 two assumptions are used in defining three phase dynamical system. The effect and meaning of them are included in topics to future research.

References

[1] M.D. Mesarovic, Y.Takahara: *Abstract Systems Theory*, (Lecture Notes in Control and Information Sciences 116), Springer, 1989.

[2] Michael Pidd: *Computer Simulation in Management Science* (Second edition), John Wiley & Sons, 1984.

[3] G.S.Fishman: *Concepts and Methods in Discrete Event Digital Simulation*, John Wiley & Sons, 1973.